The role of ontologies in creating & maintaining corporate knowledge: a case study from the aero industry

Derek Sleeman[1], Suraj Ajit[1], David W. Fowler[1], & David Knott[2]
[1]*Department of Computing Science, University of Aberdeen, Scotland, UK*
Email: {sleeman, sajit, dfowler}@csd.abdn.ac.uk
[2]*Rolls Royce plc, Derby, UK*
Email: david.knott@rolls-royce.com

Abstract. The Designers' Workbench is a system, developed to support designers in large organizations, such as Rolls-Royce, that ensures that the design is consistent with the specification for the particular design as well as with the company's design rule book(s). The evolving design is described against a jet engine ontology. Design rules are expressed as constraints over the domain ontology. Currently, to capture the constraint information, a domain expert (design engineer) has to work with a knowledge engineer to identify the constraints, and it is then the task of the knowledge engineer to encode these into the Workbench's knowledge base. This is an error prone and time consuming task. It is highly desirable to relieve the knowledge engineer of this task, and so we have developed a tool, ConEditor+ that enables domain experts themselves to capture and maintain these constraints. The tool allows the user to combine selected entities from the domain ontology with keywords and operators of a constraint language to form a constraint expression. Further, we hypothesize that to apply constraints appropriately, it is necessary to understand the context in which each constraint is applicable. We refer to this as "application conditions". We show that an explicit representation of application conditions, in a machine interpretable format, together with the constraints and the domain ontology can be used to support the verification and maintenance of constraints.

1 Introduction

The context for the principal system reported here, ConEditor+ (Ajit *et al.*, 2005; Ajit *et al.*, 2007), is the Designers' Workbench (Fowler *et al.*, 2004) that has been developed to enable a group of designers to produce cooperatively a component that conforms to the component's overall specifications and the company's design rule book(s). One can view the design rule book(s) as an important repository of corporate knowledge, in a company whose expertise is principally in the design and maintenance of aero-engines. Moreover, we are arguing that the Designers' Workbench is an interactive environment in which this corporate knowledge is applied; further, ConEditor+ allows engineers to capture and maintain (refine) these constraints (this corporate knowledge). Further, as we shall demonstrate, an ontology for describing jet engines has a central role in both these systems.

The issues faced in Knowledge Base (KB) maintenance within engineering were first raised by the XCON configuration system at Digital Equipment Corporation

(Soloway et al., 1987; Barker and O'Connor, 1989). Initially it was assumed that knowledge-based systems could be maintained by simply adding new elements or replacing existing ones. However this "simplicity" proved to be illusory as indicated by the experience of R1/XCON (Coenen, 1992).

The engineering design process has an iterative nature as designed artifacts often develop through a series of changes before a final solution is achieved. A common problem encountered during the design process is that of constraint evolution, which may involve the identification of new constraints or the modification or deletion of existing constraints. The reasons for such changes include development in the technology, changes to improve performance, changes to reduce development time and costs. In order to reduce the various maintenance problems, systems that capture and represent the rationales associated with design knowledge have been developed. Design rationales (Regli et al., 2000; Burge and Brown, 2003) capture the following types of information:

- the design alternatives considered with reasons for acceptance or rejection
- the reasons why a design decision was taken
- how certain design actions are performed

Moreover, we are interested in capturing information about when a particular design constraint is applicable. We believe it is important to know the context in which a particular constraint or a rule can be applied. We refer to this as the application condition associated with a constraint. In this paper, we present an approach that involves the explicit representation of application conditions in a machine interpretable format, together with the constraint itself. This information is used together with the appropriate domain ontology to support the verification and maintenance of constraints.

The rest of the paper is organized as follows: Section 2 provides an introduction to the Workbench, and the domain ontology used; Section 3 describes the problem(s) faced in developing the knowledge base for the Workbench and the need for ConEditor+. Section 4 gives a brief overview of ConEditor+. Section 5 then focuses on the maintenance aspects of constraints with a description of our approach. Section 6 describes how we extended the jet engine ontology and then used this in the refinement of a set of constraints from that domain. We discuss the evaluation of the approach in Section 7. Conclusions and plans for future work follow in Section 8.

2 Introduction to the Designers' Workbench

Designers in Rolls-Royce, as in many large organizations, work in teams. Thus it is important when a group of designers are working on aspects of a common project, that the subcomponent designed by one engineer is consistent with the overall specification, and with those designed by other members of the team. Additionally, all designs have to be consistent with the company's design rule book(s). Making sure that these various constraints are complied with is a complicated process, and so we have developed the Designers' Workbench, which seeks to support these activities.

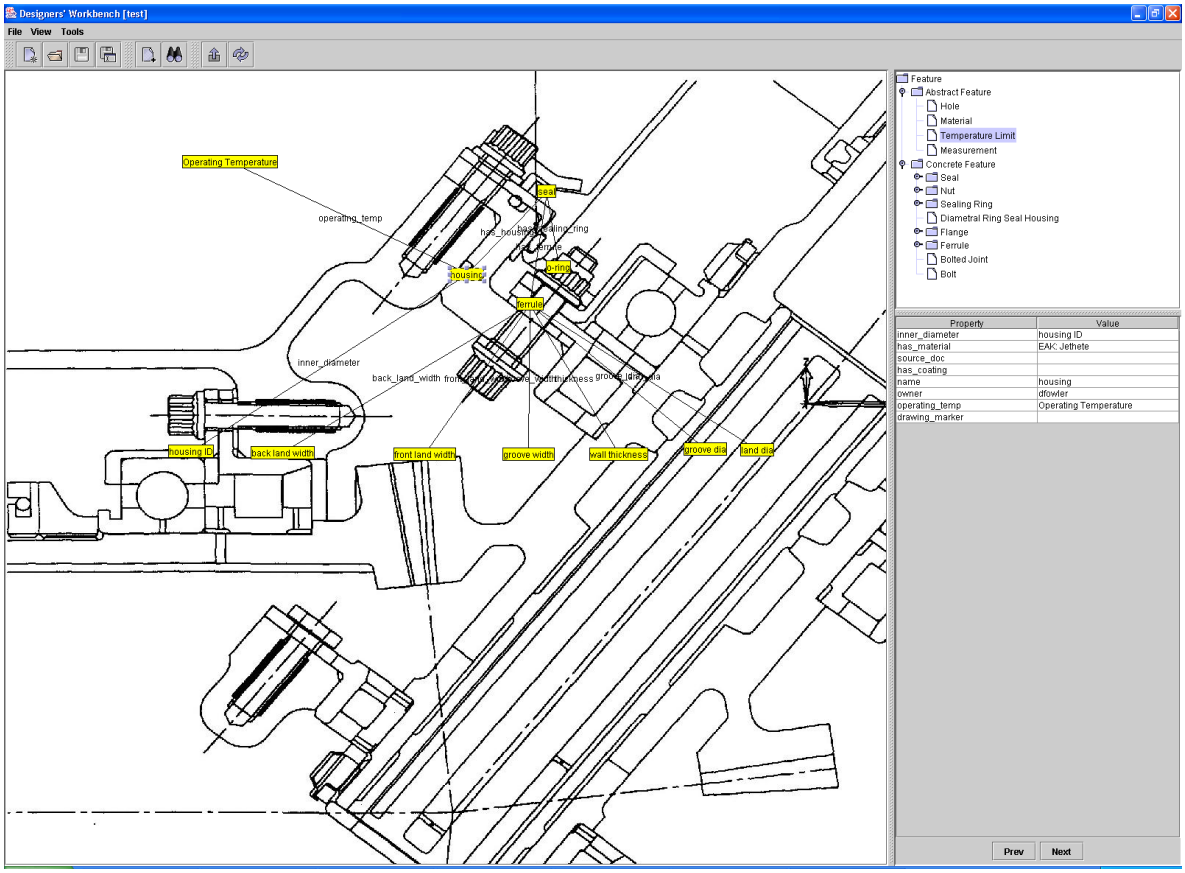

Figure 1:A screenshot of the Designers' Workbench

The Designers' Workbench (Figure 1) uses an ontology (Gruber, 1995) to describe elements in a configuration task. The system supports human designers by checking that their configurations satisfy both physical and organizational constraints. Configurations are composed of features, which can be geometric or non-geometric, physical or abstract. When a new design is input into the system an engineering drawing is provided as a graphical backcloth, and the various parts are annotated using the domain ontology. Figure 1 shows the result of such an annotation exercise; the relevant ontology displayed in the top right hand corner can be expanded to show sub-classes, properties, and relations. A graphical interface enables the designer to add new features, set property values, and perform constraint checks. If a constraint is violated, the affected features are highlighted and a report is generated. The report gives the designer a short description of the constraint that is violated, the features affected by that violation, and a link to the source document. The designer can often resolve the violations by adjusting the property values of the affected features. On selecting a feature, the GUI displays a table of corresponding properties and their values. These property values can then be adjusted, and this often resolves the constraint violation(s).

A small ontology to support the Designers' Workbench was created using the Protégé editor (Noy *et al.*, 2000), and is shown in
Figure 2. The ontology is written in OWL (McGuinness and Harmelen, 2004), and has 42 classes and 45 properties (of which 22 are object properties and 23 are datatype properties). Most classes in the ontology correspond to a type of feature, and the properties correspond to parameters that can be set for an instance of a feature (datatype properties), or to connections to other features (object properties).

Figure 2: The class hierarchy of the jet engine ontology used with the Designers' Workbench (screenshot from the OWLViz plugin for Protégé)

Figure 3 shows the properties of a class (`DiametralRingSeal`) selected from the ontology. There are three datatype properties (`in_static_joint`, `name`, and `owner`) and six object properties (`has_ferrule`, `has_housing`, `has_coating`, `has_material`, `has_sealing_ring`, and `operating_temperature`) that link to other entities in the design. Furthermore, two of the properties (`has_ferrule` and `has_housing`) are only defined for the class `DiametralRingSeal`, whereas the others are defined for classes that are ancestors of the class, and are inherited (as shown by the brackets in the screenshot).

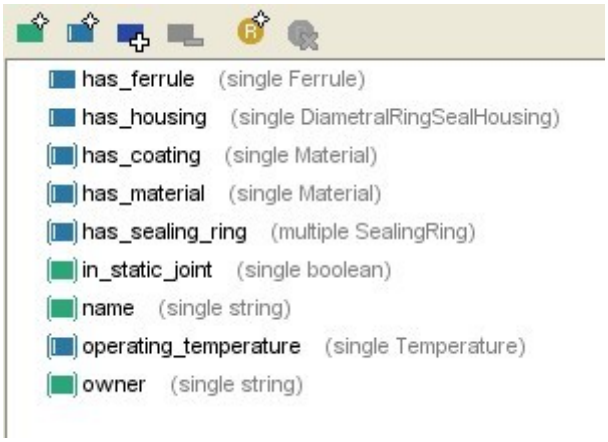

Figure 3: The properties of the class DiametralRingSeal from the jet engine ontology (screenshot from Protégé)

The values of the properties of a typical instance of the class DiametralRingSeal are shown in Figure 4. In the Designers' Workbench, property values are set by either typing values into a text box (for datatype properties), or by selecting an instance from a drop down menu (for object properties). It can be seen that values can be left uninstantiated. This enables the designer to fill in the values that are known, and to check constraints, in an incremental way.

Property	Value
name	seal_1
has_material	
has_sealing_ring	o_ring_1
has_ferrule	ferrule_1
has_coating	
in_static_joint	false
operating_temperature	
has_housing	seal_housing_1
owner	dfowler

Figure 4: The property values for a DiametralRingSeal instance (screenshot from Designers' Workbench)

3 Capturing the knowledge in the design rule books

As noted above, the Designers' Workbench needs access to the various constraints, including those inherent in the company's design rule book(s). Currently, to capture this information, a design engineer (domain expert) works with a knowledge engineer to identify the constraints, and it is then the task of the knowledge engineer to encode these into the Workbench's knowledge base. This is an error prone and time consuming task. As constraints are explained succinctly in the design rule book(s), a non-expert in the field can find it very difficult to understand the context and formulate constraints directly from the design rule book(s), and so a design engineer has to help the knowledge engineer in this process. An example of a constraint as expressed in rule

book(s) is shown in Figure 5. Adding a new constraint to the Designers' Workbench's KB currently requires coding a query in SPARQL Query Language (Prud'hommeaux and Seaborne, 2007), and a predicate in Sicstus Prolog (SICStus).

It would be useful if a new constraint could be formulated in an intuitive way, by selecting classes and properties from the appropriate ontology, and somehow combining them using a predefined set of operators. This would help engineers to input all the constraints themselves and relieve the programmer of that task. This would also enable designers to have greater control over the definition and refinement of constraints, and presumably, to have greater trust in the results of constraint checks. This led to the development of a system, known as ConEditor+, which enables a domain expert to input and maintain constraints. ConEditor+ is explained further in the next section.

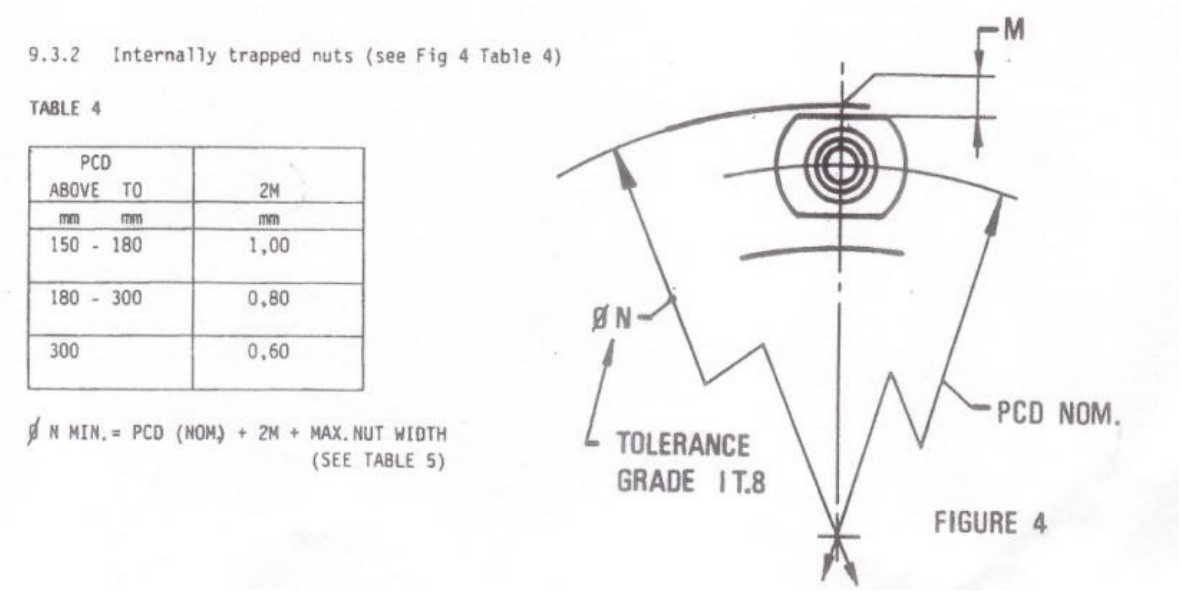

Figure 5: A constraint as expressed in a rule book

4 ConEditor+

ConEditor+ is a tool to enable domain experts to input and maintain constraints. ConEditor+'s graphical user interface (GUI) is shown in Figure 6. A constraint expression can be created by selecting entities from the taxonomy (domain ontology) and combining them with a pre-defined set of keywords and operators from the high level constraint language, CoLan (Bassiliades and Gray, 1995; Gray *et al.*, 2001). CoLan has features of both first-order logic and functional programming, and was designed to enable scientists and engineers to express constraints in a computer environment themselves.

An example of a simple constraint expressed in CoLan, against a domain ontology (a jet engine ontology) used by the Designers' Workbench is as follows:

```
constrain each f in ConcreteFeature
to have max_operating_temp(has_material(f)) >= operating_temp(f)
```

The above constraint states that for every instance of the class `ConcreteFeature`, the value of the maximum operating temperature of its material must be greater than or equal to the environmental operating temperature. The CoLan keywords used in this constraint, namely, "`constrain`", "`each`", "`in`", "`to`" and "`have`" are shown in bold.

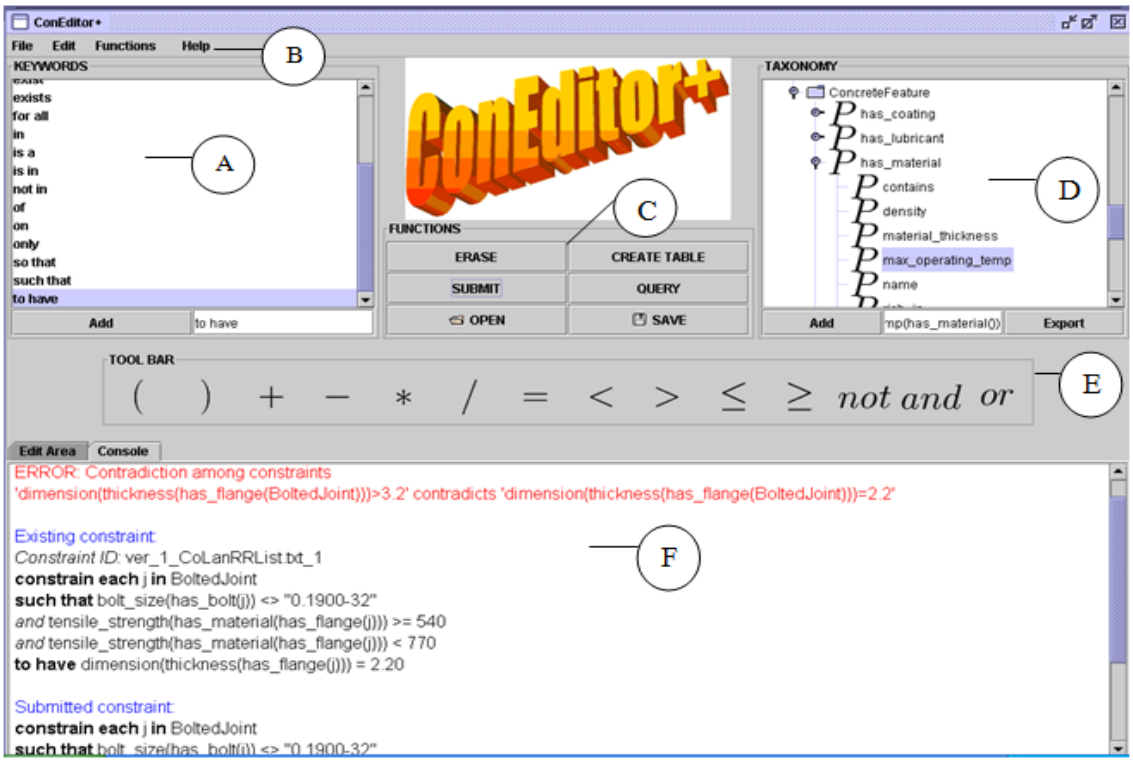

Figure 6: A screenshot of ConEditor+

ConEditor+'s GUI essentially consists of six components, namely: (A) Keywords Panel, (B) Menu Bar, (C) Functions Panel, (D) Taxonomy / Ontology Panel, (E) Tool Bar and (F) Result Panel (see Figure 6). The user can then select the appropriate entities with the mouse and so form a constraint expression. The taxonomy in the top right hand window (displayed again separately in Figure 7) shows that the class under discussion is `ConcreteFeature`, and the class `ConcreteFeature` contains the various properties `has_coating`, `has_lubricant`, `has material`, etc. Each property has a range class which, in turn, consists of more properties (e.g., `has_material` is a property that has the range class `Material`; further the class `Material` has properties `density`, `max_operating_temp`, etc.). The Taxonomy/Ontology Panel is used to select entities from the domain ontology. More details about the GUI can be found in Ajit's forthcoming thesis (Ajit, 2007). An analysis of the Rolls-Royce's design rule book(s) showed that a number of constraints are expressed in tables and so ConEditor+ provides a mechanism for inputting tables. When a constraint is modified and saved, ConEditor+ stores the modified constraint as a new version together with the original constraint. Storing all the versions would enable designers to study the evolution of constraints. Each constraint is allocated a unique identification number (ID) that includes its version number. The system provides facilities to retrieve constraints using keyword-based searches, e.g. search and retrieve all the constraints

containing the specified keyword(s) or find the constraint with the specified ID.

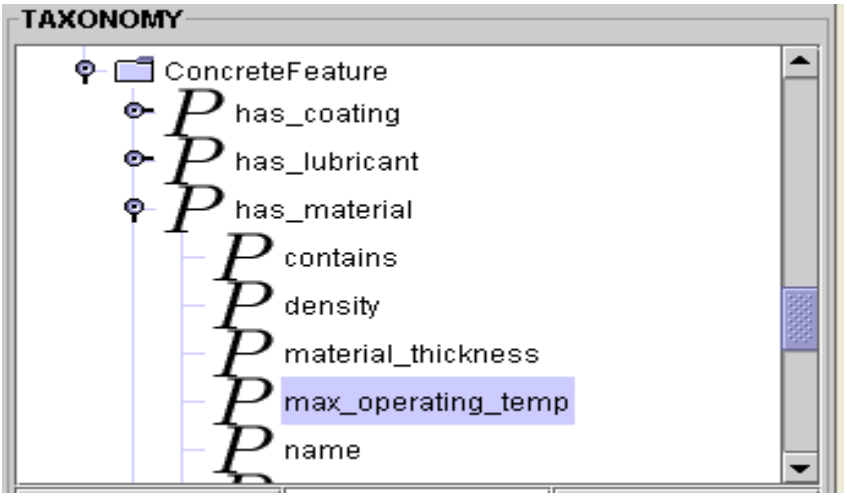

Figure 7: Taxonomy/Ontology Panel

5 Maintenance of constraints

Due to restricted availability of designers' time and for simplicity, we initially used a kite domain for our study (Yolen, 1976; Streeter, 1980; Eden, 1998) and developed an ontology for kite design. In order to explain the concept of application condition, we consider the following constraint from the kite domain together with its associated rationale and application condition:

Constraint – "The strength of the kite line needs to be greater than 90 daN[1] units."

Associated rationale – "This provides the required stability for the kite to fly."

Application condition – "This is applicable to stunt kites of standard size in strong winds only."

The difference between a rationale and an application condition is evident from the example considered above; the rationale states the reason for a constraint (why), whereas the application condition states the context in which it is applicable (when).

In order to tackle the various maintenance issues, our approach has the following stages:

- Capture the "context" of a constraint, in a machine interpretable form, as an application condition associated with the constraint.
- Use the application condition together with the constraint and the appropriate domain ontology to support verification and maintenance

We have extended ConEditor+ so that the user (the domain expert) can associate an

[1] decaNewton, a common metric unit of force.

application condition with each of the constraints. Often, such information is implicit to the person who formulates the constraint. We believe that it is important to make the application condition explicit so that it can be used for both verification and maintenance. The assumptions on which a constraint is based may no longer be true and in such cases, it becomes necessary to deactivate or remove those constraints from the KB. Further, an application condition may not be relevant to a particular design task.

ConEditor+ captures both the constraints and the application conditions in the same language, CoLan. Both the constraints and the application conditions are then automatically converted into a standard machine interpretable format known as Constraint Interchange Format (CIF) (Gray *et al.*, 2001). We give below a typical constraint and its application condition in CoLan:

```
constrain each k in Kite
such that has_type(k) = "Flat" and has_shape(k) = "Diamond"
to have tail_length(has_tail(k)) = 7 * spine_length(has_spine(k))
```

In the above constraint, the application condition (in italics) is introduced by the clause "such that". This constraint states that the length of a tail of a kite needs to be seven times the length of the spine of the kite; however, this constraint is only applicable to flat diamond-shaped kites.

In order to make it clearer, we divide a CoLan, into three parts namely antecedent, application condition and consequent. Thus, the above constraint consists of:

Antecedent: `constrain each k in Kite`

Application condition: `such that has_type(k) = "Flat" and has_shape(k) = "Diamond"`

Consequent: `tail_length(has_tail(k)) = 7 * spine_length(has_spine(k))`

6 Extension of Jet Engine Ontology and Maintenance of a more Complex Set of Constraints

After a successful application and evaluation of ConEditor+ in the domain of kite design (for more information on this work see (Ajit, 2007)), we decided to apply our approach to part of the considerably more demanding Rolls-Royce domain. We initially reviewed, in some detail, the ontology used to support Designers' Workbench, and then analyzed a considerable number (72) of additional Rolls-Royce's design standard documents which contain rules/standards for the design of various parts and processes involved in civil aero-engines. Interviews were held with a design engineer at Rolls-Royce, Derby. We then extended the jet engine ontology to incorporate the additional information (e.g. concepts) obtained from these analyses. Figure 8 shows a screenshot of the extended jet engine ontology developed using Protégé (Noy et al., 2000). We then expressed all the constraints together with their application conditions against the extended jet engine ontology. There are a number of ways in which we can use the domain ontology together with the constraints and application conditions to support the

verification and maintenance of constraints. Refinement of the constraint KB is described, in some detail, below.

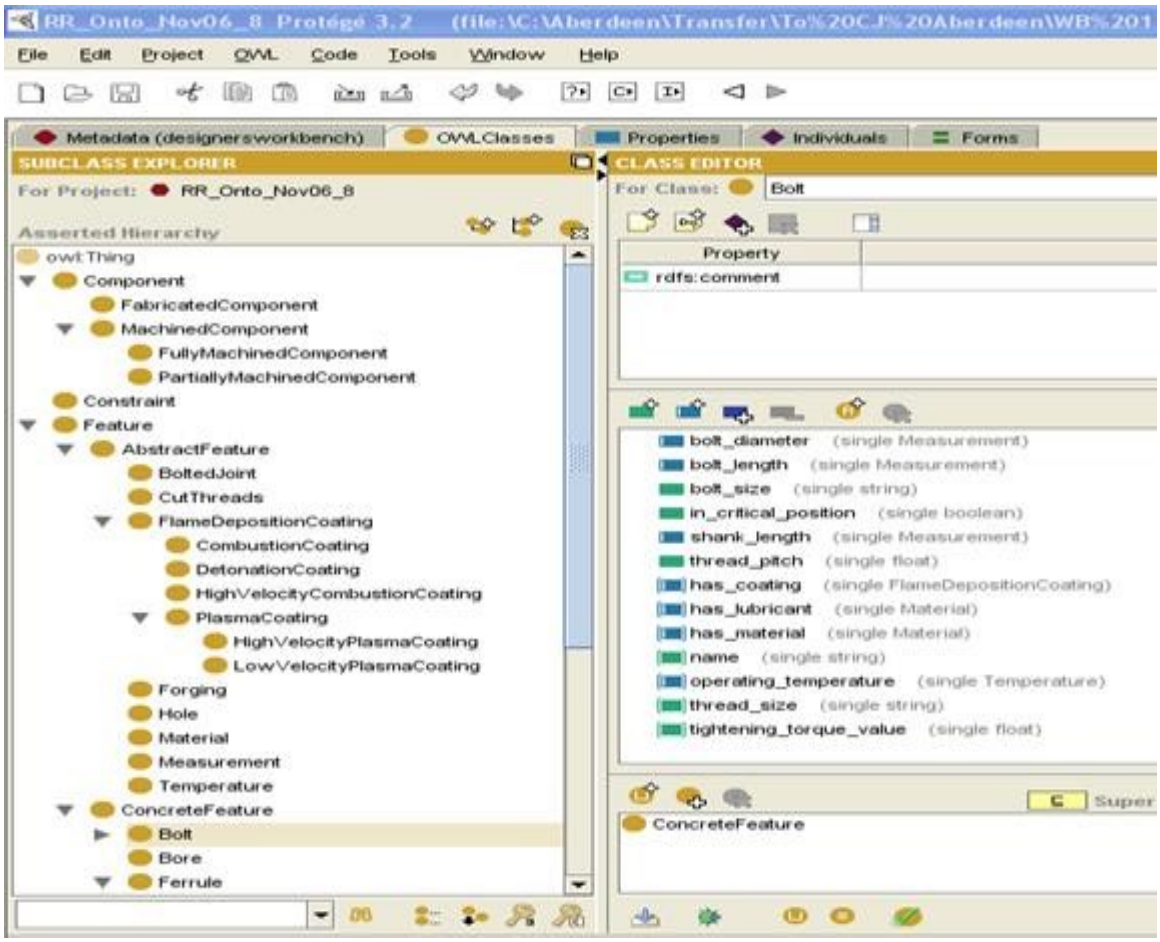

Figure 8: Ontology of a part of the Rolls-Royce domain in Protégé

6.1 Redundancy

Redundancy occurs between constraints when all the components of a constraint (antecedent, application condition and consequent) are equivalent to the corresponding components of another constraint. An example of redundancy is:

```
(i)constrain each c in FlameDepositionCoating
such that has_fabricated_component(c)
and has_mask_location_level(c) = "difficult"
to have has_max_overspray_thickness(c) = 4.0

(ii)constrain each c in FlameDepositionCoating
such that has_fabricated_part(c)
and has_mask_location_level(c) = "difficult"
to have has_max_overspray_thickness(c) = 4.0
```

As `has_fabricated_component` is an equivalent property to `has_fabricated_part` in the domain ontology one can infer that the constraint (i) is equivalent to constraint (ii). The domain expert is notified of this by ConEditor+, which

suggests the redundancy be eliminated.

6.2 Subsumption

Subsumption occurs between a pair of constraints when one constraint "covers" all the conditions of another constraint i.e. constraint A subsumes constraint B, if B is satisfied whenever A is satisfied. An example of this type of subsumption follows:

```
(iii)constrain each s in RingSeal
such that has_elastometric_toroidal_oring(s) and
name(has_material(has_sealing_ring(s))) <> "perfluorocarbon" and
pressure_type_hou_mat_flange(s) = "internal"
to have min_face_groove_dia(s) = max_face_groove_dia(s) - 0.25

(iv)constrain each s in FaceRingSeal
such that has_elastometric_toroidal_oring(s)
and name(has_material(has_sealing_ring(s))) <> "perfluorocarbon"
and pressure_type_hou_mat_flange(s) = "internal"
to have min_face_groove_dia(s) = max_face_groove_dia(s) - 0.25
```

As `FaceRingSeal` is a subclass of `RingSeal` in the domain ontology one can infer that the constraint (iii) subsumes constraint (iv). The domain expert is notified of this fact, and ConEditor+ suggests the expert removes / deactivates constraint (iv). (It is then left to the domain expert to take the appropriate action.)

6.3 Inconsistency/Contradiction

An inconsistency/contradiction occurs between a pair of constraints when the consequent of one constraint contradicts the consequent of another constraint while the antecedents and application conditions are equivalent i.e., constraint A contradicts constraint B or vice-versa if both constraints A and B are unsatisfiable. An example of this type of inconsistency follows:

```
(v)constrain each c in Component
such that name(component_coating(c)) = "silver"
and name(component_material(c)) = "steel"
to have tensile_strength(component_material(c)) < 1390

(vi)constrain each c in Component
such that name(component_coating(c)) = "silver"
and name(component_material(c)) = "steel"
to have tensile_strength(component_material(c)) > 1590
```

By comparing the two constraints above, one can infer that the constraint (v) contradicts constraint (vi). The domain expert is notified of this by ConEditor+ which suggests the domain expert takes an appropriate action (modify/delete).

6.4 Overview

ConEditor+ can also deal with the situation where a constraint needs to have multiple refinements applied before it is possible to determine whether another constraint is equivalent / subsumed / inconsistent. For examples see (Ajit, in preparation).

6.5 Implementation

ConEditor+ is implemented in the Java programming language; the domain ontology in OWL (McGuinness and Harmelen, 2004) is developed using the Protégé ontology editor (Noy *et al.*, 2000) and accessed using Jena (HP, 2000). Any syntactic errors among constraints are detected by ConEditor+ with the help of a Daplex compiler (Bassiliades and Gray, 1995). The constraints are initially expressed in CoLan and then converted into a standard Constraint Interchange Format (CIF) (Gray *et al.*, 2001) using a translator. ConEditor+ uses the domain ontology together with the CIF representation of constraints and application conditions to detect the various refinements between pairs of constraints, and thus helps the domain expert with the verification and maintenance of constraints. The domain ontology plays an important role in detecting refinements together with the application conditions.

7 Evaluation

Before adding the application condition feature, we performed an evaluation of ConEditor+. A demonstration was given to a group of design engineers at Rolls-Royce. The demonstration involved the following three stages:

a) Presenting the constraint as found in the rule book i.e. as a mixture of textual and graphical information (Figure 5)
b) Presenting the subjects with the same information but expressed this time in CoLan.
c) Demonstrating to the subjects how the CoLan expression could be captured by ConEditor+. (ConEditor+ was, in fact, used by the experimenter, Suraj Ajit).

The design engineers were able to follow the three stages. They found the GUI simple, user-friendly and fairly intuitive to use. However they felt they would need some training before they could perform the last two stages [(b) and (c)] unsupported. They also made the general point that their company has a Design Standards group that is responsible for creating and maintaining the company-wide rule book(s). They would expect this group to use systems such as ConEditor+ to input constraints. The designers would then subsequently use the information either in the current form or in the Designers' Workbench-like environment (see Figure 9).

After implementing the application condition feature, we conducted several further experiments including:

We demonstrated ConEditor+'s features to five subjects (two mechanical engineering research students, two computer science research students and one computer science research fellow). Each subject was then given the task of inputting a set of constraints in CoLan using ConEditor+. The subjects were asked to use ConEditor+ to resolve inconsistencies (contradictions) and also to follow suggestion(s)

given by ConEditor+ to refine (fuse, eliminate redundancies and subsumptions) the constraints and their associated application conditions. A questionnaire about the usability of ConEditor+ and its maintenance features was given to the subjects, who were asked to use a 5 point rating scale (1 being poor and 5 being excellent).

Results: All the subjects found ConEditor+ fairly easy to use and helpful for the verification and maintenance of constraints. The average overall rating given by the subjects, for both the usability and maintenance features of ConEditor+ was 3.8. Additionally, some subjects gave helpful suggestions for improving the usability of ConEditor+.

8 Conclusions and Future Work

This paper describes a methodology to enable domain experts to capture and maintain constraints in an engineering design environment. An ontology is used to represent the domain knowledge and constraints are expressed against this ontology. The context is a system known as the Designers' Workbench that has been developed to automatically check if all the constraints have been satisfied and if not, enable the designers to resolve them. To function, the Designers' Workbench must be provided with a set of task specific requirements, and generic (company-wide) design constraints. Originally, the latter needs a knowledge engineer to study the design rule book(s), consult the design engineer (domain expert) and encode all the constraints into the Designers' Workbench's KB. We described the tool ConEditor+ that has been developed to help domain experts themselves capture and maintain engineering design constraints.

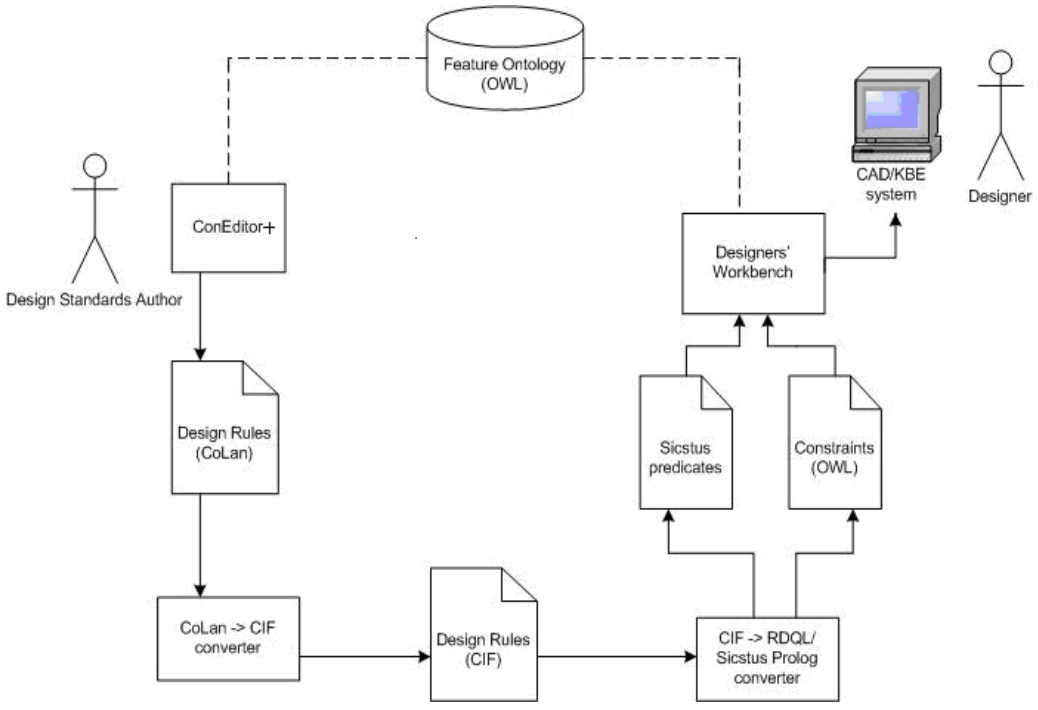

Figure 9: Proposed system architecture

We hypothesize that to apply constraints appropriately, it is necessary to capture the contexts (application conditions) associated with the constraints and that these,

together with the appropriate domain ontology, could be used for verification and maintenance. On the basis of the studies done in the domain of kite design and then in part of the Rolls-Royce domain, we believe the above hypothesis to be true; we also believe ConEditor+ is a useful tool for design engineers to capture and maintain constraints.

As part of future work, we plan to complete the implementation of the proposed architecture (Figure 9) that shows how ConEditor+ fits into a wider framework. A Design Standards author initially inputs all the design rules (constraints) into ConEditor+. The design constraints are then converted into a standard machine interpretable format (CIF) and processed by the Designers' Workbench. As can be seen from Figure 9, it is planned to interface the Designers' Workbench to a sophisticated knowledge-based engineering (KBE) system. The designers' workbench will be called from the main system, the KBE, effectively as a sub-process to check the consistency of a design, or part of a design, produced by the KBE. Additionally, it is desirable for experienced designers to be able to indicate when constraints or application conditions need modifying as they are inconsistent with their experience. Such modifications of the corporate knowledge need to be done consistently if such KBs are to capture the company's "cutting edge" knowledge.

In fact, Figure 9 only represents one aspect (the design rule book) of the knowledge which is both generated and used in a contemporary knowledge-based engineering firm which is involved in design, manufacturing & maintenance. For example, there are a number of further additional knowledge repositories needed by today's KBE systems, including:

- Design templates (and conditions under which they should be used, i.e. *application conditions*)
- Libraries of designs for components and their rationales
- Requirements and constraints of the various manufacturing environments
- Best practices as collected by several parts of the organization (including designers)
- Requirements and constraints mandated by the several organizations which service the engines
- Feedback from the servicing and maintenance organizations which indicate which problems actually arise in the field, some analysis of their possible causes, and suggested remedies.

The latter type of information is the focus of the IPAS project (www.3worlds.org), a DTI / Rolls-Royce funded project, which started in 2005. The last source of data quoted above makes it clear that there is an important Information Life Cycle inherent in the aero- industry, where information flows from Design to the Manufacturing units, and then to the Service / Maintenance facilities; the later in turn creates information which needs to be passed to designers so that future engines can be improved as a result of real-world feedback. Figure 10 shows this cycle:

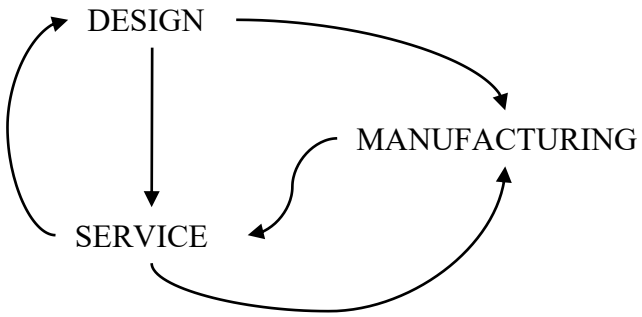

Figure 10: The principal Information Life Cycle in the Aero-Industry

It is also clear that there are vast amounts of data and information available from a variety of sources, and to make this information inter-operational, there is potentially a major role for ontologies as many of the data / information sources use different terminologies. This is certainly an important role for ontologies in the IPAS project. In fact in both the projects undertaken with Rolls-Royce (AKT and IPAS) we are not only using standard ontology maintenance procedures, but we are encountering many of the problems of contemporary ontology engineering, namely:

- ontology creation (seeking to develop ontologies systematically and to ensure that relevant aspects of trust and provenance are captured; deciding whether or not domain ontologies should be developed from high-level ontologies;
- ontology evolution (an ontology developed for one engine may need to be modified so that it is applicable to a future engine) and,
- ontology modularization (for some services a sparse description of, say, the combustion chamber may be sufficient, but for other services much greater detail may be required).

Acknowledgements

This work is supported by the EPSRC Sponsored Advanced Knowledge Technologies project, GR/NI5764, which is an Interdisciplinary Research Collaboration involving the University of Aberdeen, the University of Edinburgh, the Open University, the University of Sheffield and the University of Southampton. We would like to acknowledge the assistance of engineers and designers in the Transmissions and Structures division of Rolls-Royce plc, Derby, UK. The IPAS project is funded by Rolls-Royce and the Department of Trade and Industry under the Technology Program, DTI Reference TP/2/IC/6/I/10292.

References

Ajit, S. (in preparation). Capture and Maintenance of Constraints in Engineering Design. PhD Thesis, University of Aberdeen, Aberdeen, UK.

Ajit, S., Sleeman, D., Fowler, D. W., Knott, D. and Hui, K. (2005). Acquisition and Maintenance of Constraints in Engineering Design. *Proceedings of the 3rd International Conference on Knowledge Capture, KCAP 2005*, 173-174.

Ajit, S., Sleeman, D., Fowler, D. W., Knott, D. and Hui, K. (2007). ConEditor+: Capture and Maintenance of Constraints in Engineering Design. *Proceedings of the IJCAI-07 Workshop on "Knowledge Management & Organizational Memories"*, 6-11.

Barker, V. E. and O'Connor, D. E. (1989). Expert Systems for Configuration at Digital: XCON and Beyond. *Communications of the ACM*, 32 (3), 298-318.

Bassiliades, N. and Gray, P. (1995). CoLan: A Functional Constraint Language and Its Implementation. *Data and Knowledge Engineering*, 14 (3), 203-249.

Burge, J. and Brown, D. C. (2003). Rationale Support for Maintenance of Large Scale Systems. *Workshop on Evolution of Large-Scale Industrial Software Applications (ELISA), ICSM '03*.

Coenen, F. P. (1992). A Methodology for the Maintenance of Knowledge based Systems. *Niku-Lari, A. (Ed), EXPERSYS-92 (Proceedings), IITT-International*, 171-176.

Eden, M. (1998). The Magnificient Book of Kites: Explorations in Design, Construction, Enjoyment and Flight. Black Dog & Levanthal Publishers, New York.

Fowler, D. W., Sleeman, D., Wills, G., Lyon, T. and Knott, D. (2004). Designers' Workbench. *Proceedings of the Twenty-fourth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, 209-221.

Gray, P., Hui, K. and Preece, A. (2001). An Expressive Constraint Language for Semantic Web Applications. *E-Business and the Intelligent Web: Papers from the IJCAI-01 Workshop*, 46-53.

Gruber, T. R. (1995). Towards Principles for the Design of Ontologies Used for for Knowledge Sharing. *International Journal of Human-Computer Studies*, 43 (5-6), 907-928.

HP (2000). Helwett Packard Labs, Jena - A Semantic Web Framework for Java. [online]. Available from: http://jena.sourceforge.net/ [Accessed 27 July 2007].

McGuinness, D. L. and Harmelen, F. v. (2004). OWL Web Ontology Language Overview, W3C Recommendation 10 February 2004. [online]. Available from: http://www.w3.org/TR/owl-features/ [Accessed 27 July 2007].

Noy, N. F., Fergerson, R. W. and Musen, M. A. (2000). The knowledge model of Protege-2000: Combining interoperability and flexibility. *International Conference on Knowledge Engineering and Knowledge Management (EKAW' 2000)*.

Prud'hommeaux, E. and Seaborne, A. (2007). SPARQL Query Language for RDF, W3C Working Draft 26 March 2007. [online]. Available from: http://www.w3.org/TR/rdf-sparql-query/ [Accessed 27 July 2007].

Regli, W. C., Hu, X., Atwood, M. and Sun, W. (2000). A Survey of Design Rationale Systems: Approaches, Representation, Capture and Retrieval. *Engineering with Computers: An Int'l Journal for Simulation-Based Engineering, special issue on Computer Aided Engineering in Honor of Professor Steven J. Fenves*, 16, 209-235.

SICStus. Swedish Institute of Computer Science. [online]. Available from: http://www.sics.se/sicstus/ [Accessed 27 July 2007].

Soloway, E., Bachant, J. and Jensen, K. (1987). Assessing the Maintainability of XCON-in-RIME: Coping with Problems of a Very Large Rule-Base. *Proceedings of AAAI-87*, 824-829.

Streeter, T. (1980). The Art of the Japanese Kite. Tokyo: Charles E Tuttle Company Inc.

Yolen, W. (1976). The Complete Book of Kites and Kite Flying. New York: Simon and Schuster Trade.